Numerical Optimization and the Toolkit for Advanced Optimization

Jason Sarich, Todd Munson, Jorge Moré

Mathematics and Computer Science Division,
Argonne National Laboratory

August 19, 2009

Part I

Nonlinear Optimization

Nonlinear Optimization

- Unconstrained Optimization
- Bound-constrained Optimization
- General Constrained Optimization

Nonlinear Optimization

Unconstrained Optimization Problem

$$f: \mathbb{R}^N \mapsto \mathbb{R}$$
$$\min_{x \in \mathbb{R}^N} f(x)$$

Nonlinear Optimization

Bound-constrained Optimization Problem

$$\begin{array}{rcl} \min & f(x) & \text{(objective function)} \\ \text{subject to} & x_l \leq x \leq x_u & \text{(bounds)} \end{array}$$

Nonlinear Optimization

Constrained Optimization Problem

$$\begin{array}{rcl} \min & f(x) & \text{(objective function)} \\ \text{subject to} & c_l \leq c(x) \leq c_u & \text{(constraints)} \end{array}$$

Note: TAO is not able to solve constrained optimization problems directly.

Part II

Algorithms

Algorithms

Nonlinear optimization algorithms are iterative processes. In many cases, each iteration involve calculating a 'search direction', then function values along that direction are calculated until certain conditions are met.

- Newton's Method
- Quasi-Newton Methods
- Conjugate Gradient

Algorithms

Newton's Method

- Step 0 Choose initial vector x_0
- Step 1 Compute gradient $\nabla f(x_k)$ and Hessian $\nabla^2 f(x_k)$
- Step 2 Calculate the direction d_{k+1} by solving the system:

$$\nabla^2 f(x_k) d_{k+1} = -\nabla f(x_k)$$

- Step 3 Apply line search algorithm to obtain "acceptable" new vector:

$$x_{k+1} = x_k + \tau d_{k+1}$$

- Return to Step 1

Algorithms

Problems with Newton's Method

- Hessian must be derived, computed, and stored
- Linear solve must be performed on Hessian

Algorithms

Quasi-Newton Methods

Use approximate Hessian $B_k \approx \nabla^2 f(x_k)$. Choose a formula for B_k so that:

- B_k relies on first derivative information only
- B_k can be easily stored
- $B_k d_{k+1} = -\nabla f(x_k)$ can be easily solved

Algorithms

Conjugate Gradient Algorithms

These algorithms are an extension of the conjugate gradient methods for solving linear systems.

$$d_{k+1} = -\boldsymbol{\nabla} f(x_k) + \beta_k d_k$$

Some possible choices of β_k ($g_k = \nabla f(x_k)$):

$$\beta_k^{FR} = \left(\frac{\|g_{k+1}\|}{\|g_k\|} \right)^2, \qquad \text{Fletcher-Reeves}$$

$$\beta_k^{PR} = \frac{\langle g_{k+1}, g_{k+1} - g_k \rangle}{\|g_k\|^2}, \qquad \text{Polak-Ribière}$$

$$\beta_k^{PR+} = \max\left\{ \beta_k^{PR}, 0 \right\}, \qquad \text{PR-plus}$$

Algorithms

Derivate Free Algorithms

There are some applications for which it is not feasible to find the derivative of the objective function. There are some algorithms available that can solve these applications, but they can be very slow to converge.

- Pattern Searches
- Nelder-Mead Simplex
- Model-based methods
- Use finite differences

Part III

TAO

The process of nature by which all things change and which is to be followed for a life of harmony

TAO

What does TAO do for you?

- Contains a library of optimization solvers for solving unconstrained, bound-constrained, and complementarity optimization problems. These solvers include Newton methods, Quasi-Newton methods, conjugate gradients, derivative free, and semi-smooth methods.
- Provides C, C++, and Fortran interfaces to these libraries
- Allows for large scale, sparse objects, and parallel applications
- Uses PETSc data structures and utilities

TAO Solvers

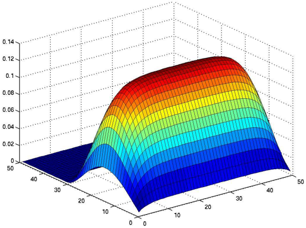
	handles bounds	requires gradient	requires Hessian
lmvm	no	yes	no
nls	no	yes	yes
ntr	no	yes	yes
ntl	no	yes	yes
cg	no	yes	no
nm	no	no	no
blmvm	yes	yes	no
tron	yes	yes	yes
gpcg	yes	yes	no

TAO

Pressure in a Journal Bearing

$$\min \left\{ \int_{\mathcal{D}} \left\{ \tfrac{1}{2} w_q(x) \|\nabla v(x)\|^2 - w_l(x) v(x) \right\} \, dx : v \geq 0 \right\}$$

$$w_q(\xi_1, \xi_2) = (1 + \epsilon \cos \xi_1)^3$$
$$w_l(\xi_1, \xi_2) = \epsilon \sin \xi_1$$
$$\mathcal{D} = (0, 2\pi) \times (0, 2b)$$

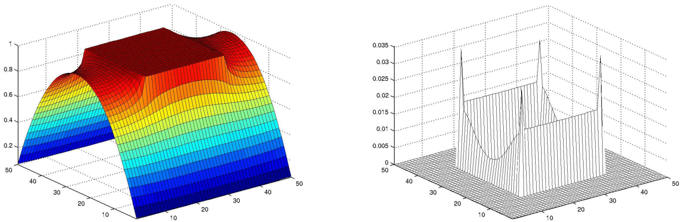


Number of active constraints depends on the choice of ϵ in $(0, 1)$.
Nearly degenerate problem. Solution $v \notin C^2$.

TAO

Minimal Surface with Obstacles

$$\min\left\{\int_{\mathcal{D}}\sqrt{1+\|\boldsymbol{\nabla} v(x)\|^2}\,dx : v \geq v_L\right\}$$



Number of active constraints depends on the height of the obstacle. The solution $v \notin C^1$. Almost all multipliers are zero.

TAO

Parallel Performance

Processors Used	BLMVM Iterations	Execution Time	Percentage of Time AXPY	Dot	FG
8	996	1083.8	31	9	60
16	991	538.2	30	10	60
32	966	267.7	29	11	60
64	993	139.5	27	13	60
128	987	72.4	25	15	60
256	996	39.2	26	18	56
512	1000	21.6	23	22	53

Table: Scalability of BLMVM on Obstacle Problem with 2,560,000 variables.

TAO

What TAO doesn't do

- Application Modeling
- Derivatives
- Linear programming
- Constrained optimization
- Integer programming
- Global minimization

TAO Applications

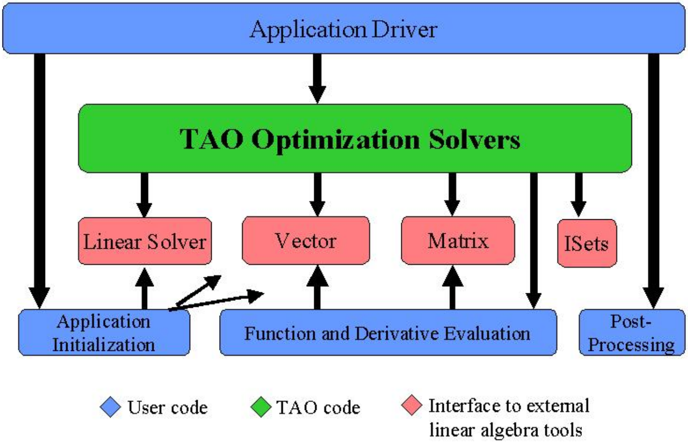
Using TAO

There are two parts to solving an optimization application with TAO:

- An **Application Object** that contain routines to evaluate an objective function, define constraints on the variables, and provide derivative information.
- A driver program (`main`) that creates a **TAO solver** with desired algorithmic options and tolerances and connects with the application object.

By default, TAO uses Matrix, Vector, and KSP objects from PETSc but can be extended to other linear algebra packages.

TAO Application



TAO Applications

What do you need to do for the Application Object?

You need to write C, C++, or Fortran functions that:

- Set the initial variable vector
- Compute the objective function value at a given vector
- Compute the gradient at a given vector
- Compute the Hessian matrix at a given vector (for Newton methods)
- Set the variable bounds (for bounded optimization)

TAO Applications

Create a data structure that contains any state information, such as parameter values or data viewers, that the evaluation routines will need. For example:

```
typedef struct {
    double      epsilon; /* application parameter */
    PetscViewer pv;      /* helpful for debugging */
} UserContext;
```

The objective function evaluation routine should look like:

```
int MyFunction(TAO_APPLICATION app, Vec x,
                     double *fcn, void *userCtx){
   UserContext *user = (UserContext *)userCtx;
   ...
}
```

TAO Applications

The routines for computing the gradient and Hessians look similar:

```
int MyGradient(TAO_APPLICATION app, Vec x, Vec g,
                     void *userCtx){
   UserContext *user = (UserContext *)userCtx;
   ...
}
int MyHessian(TAO_APPLICATION app, Vec x, Mat *H,
                Mat *Hpre, int *flag, void *userCtx){
   UserContext *user = (UserContext *)userCtx;
   ...
}
```

TAO Applications

Writing the Driver

A “driver” program is used to hook up the user’s application to the TAO library. This driver performs the following steps:

- Create the TAO Solver and Application objects
- Create the variable vector and Hessian matrix
- Hook up the Application to TAO
- Solve the application

TAO Applications

Create the TAO Solver and Application objects

```
TAO_SOLVER       tao;  /* TAO Optimization solver     */
TAO_APPLICATION  app;  /* TAO Application using PETSc */
UserContext      user; /* user-defined structure      */
Vec              x;    /* solution vector             */
Mat              H;    /* Hessian Matrix              */

PetscInitizialize(&argc,&argv,0,0);
TaoInitialize(&argc,&argv,0,0);
TaoCreate(PETSC_COMM_SELF,"tao_lmvm",&tao);
TaoApplicationCreate(PETSC_COMM_SELF,&app);
...
```

TAO Applications

Create storage for the solution vector and Hessian matrix

```
TAO_SOLVER      tao; /* TAO Optimization solver     */
TAO_APPLICATION app; /* TAO Application using PETSc */
UserContext     user;/* user-defined structure      */
Vec             x;   /* solution vector             */
Mat             H;   /* Hessian Matrix              */

...
VecCreateSeq(PETSC_COMM_SELF,n,&x);
MatCreateSeqAIJ(PETSC_COMM_SELF,n,n,nz,PETSC_NULL,&H);
...
```

TAO Applications

Hook up the application to TAO

```
TAO_SOLVER      tao;  /* TAO Optimization solver     */
TAO_APPLICATION app;  /* TAO Application using PETSc */
UserContext     user; /* user-defined structure      */
Vec             x;    /* solution vector             */
Mat             H;    /* Hessian Matrix              */

...
user.epsilon = 0.1;
TaoAppSetInitialSolutionVec(app,x);
TaoAppSetObjectiveRoutine(app,MyFunction,(void *)&user);
TaoAppSetGradientRoutine(app,MyGradient,(void *)&user);
TaoAppSetHessianRoutine(app,MyHessian,(void *)&user);
...
```

TAO Applications

Solve the application

```
TAO_SOLVER       tao;  /* TAO Optimization solver      */
TAO_APPLICATION  app;  /* TAO Application using PETSc  */
UserContext      user; /* user-defined structure       */
Vec              x;    /* solution vector              */
Mat              H;    /* Hessian Matrix               */

...
TaoSolveApplication(app, tao);
VecView(x,PETSC_VIEWER_STDOUT_SELF);
```

TAO Applications

Solve a multiple processor application

The most important and difficult part of solving a multiple processor application is writing the function, gradient, and Hessian evaluation routines to run in parallel.
Once that is done, it is trivial to get TAO to run in parallel:

```
...
TaoCreate(PETSC_COMM_WORLD,"tao_lmvm",&tao);
TaoApplicationCreate(PETSC_COMM_WORLD,&app);
VecCreateMPI(PETSC_COMM_WORLD,n,&x);
MatCreateMPIAIJ(PETSC_COMM_WORLD,n,n,nz,PETSC_NULL,&H);
...
```

Toolkit for Advanced Optimization

- You can download TAO from the webpage
- The documention online includes installation instructions, a user's manual and a man page for every TAO function.
- The download includes several examples for using TAO in C and Fortran. We will use some of these examples in the tutorial.
- If you have any questions, please contact us at